Type inference for Core Erlang [1]

Gábor Oláh, Dániel Horpácsi and Melinda Tóth

Eötvös Loránd University & ELTE-Soft Nonprofit Ltd

`{olikas,daniel-h,tothmelinda}@elte.hu`

Erlang [2] is a dynamically typed, functional, concurrent programming language. *Core Erlang* [4] is a pure functional variant of the base language, where any construct of Erlang can be easily expressed in Core Erlang, whilst preserving most of the static semantic properties, like types.

In Erlang, types of variables and functions are not defined in the program. Although the compiler performs some strict type checks (i.e. no implicit type conversions can happen), if a function (or an operator) is invoked with an unexpected type of data, only a run-time exception is thrown. Such programming errors are a tedious task to reveal, therefore several tools have been made to help programmers finding possible discrepancies in the program, i.e. situations where type mismatch can happen. However, the language allows polymorphic return types for branching expressions, which makes the type system very complex, and the types uneasy to comprehend. In order to overcome this issue, *success typing* [5] has been introduced, reducing the aforementioned complexity by substituting an upper bound type for complex union types and making types readable; on the other hand, it loses static type information.

Success typing become the de-facto type inference algorithm for Erlang. It yields an overapproximation of types, which increases readability, but decreases accuracy. We cannot use success types, for example, for test data generation, since they are too general, so that the data we get based on success typing will likely be improperly typed. Our goal is to make a type inference system for Erlang that derives types accurate enough for test data generation. In particular, we transform these types to *QuickCheck* [3] data generators that can supply functions with random arguments, which we utilize to build an Erlang benchmarking system.

By introducing a new, more precise type system to Core Erlang, the information loss can be decreased for the price of long type expressions. As stated already, Erlang programs can be easily turned into static semantically equivalent Core Erlang programs, where the functions preserve their types. The long, but way more accurate types are more suitable for random argument generation. We define type inference rules and the full algorithm is described. A comparison of our achievements with earlier results is provided, as well as a proposal to extend the results for the full Erlang language is given.

[1]This work has been supported by the European Union Framework 7 under contract no. 288570 "ParaPhrase: Parallel Patterns for Adaptive Heterogeneous Multi-core System", `http://paraphrase-enlarged.elte.hu`.

References

[1] A. Aiken, E. L. Wimmers, and T. K. Lakshman. Soft Typing with Conditional Types. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, pages 163–173, New York, NY, USA, 1994. ACM.

[2] J. Armstrong. *Programming Erlang: Software for a Concurrent World.* Pragmatic Bookshelf, 2007.

[3] T. Arts, J. Hughes, J. Johansson, and U. Wiger. Testing Telecoms Software with Quviq QuickCheck. In *Proceedings of the 2006 ACM SIGPLAN Workshop on Erlang*, ERLANG '06, pages 2–10, New York, NY, USA, 2006. ACM.

[4] R. Carlsson. An introduction to Core Erlang. In *In Proceedings of the PLI01 Erlang Workshop*, 2001.

[5] T. Lindahl and K. Sagonas. Practical Type Inference Based on Success Typings. In *Proceedings of the 8th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '06, pages 167–178, New York, NY, USA, 2006. ACM.